Compressed Storage of TRANSIMS Plans

B. W. Bush
Los Alamos National Laboratory
6 February 1997

Abstract

There exist numerous methods for storing TRANSIMS plan data. This presentation outlines some of them and compares their storage requirements for a real-life plan set containing nearly 300,000 plans. We find that practical methods of plan compression can significantly reduce the amount of storage (in memory or on disk) needed to store plans.

Outline

- *motivation*
- *information-theoretic approach*
- *storage methods*
  - *link ID (ASCII)*
  - *link ID (binary)*
  - *unique link ID (binary)*
  - *run-length encoding of link ID (ASCII)*
  - *turn (ASCII)*
  - *turn (binary)*
  - *Huffman turn compression (binary)*
  - *other*
- *comparison study*
- *summary comparison of methods*
- *future work*

Motivation

- *Over the past two years, there has been much discussion within the TRANSIMS team concerning how plans can be stored efficiently.*
- *Plan data currently requires a lot of storage, whether on disk or in memory.*
  - *It takes 100-250 MB (depending on file format) to store the ~300,000 plans used in the DFW case study; these represent the plans that start between 5AM and 10AM and pass through the 25 square mile study area.*
  - *There are 5,886,500 person-trips per day in the Portland area.*
- *Information-theoretic techniques can be applied to plan sets to understand the redundancy in and other characteristics of plans.*

[For now, we only consider the storage of trajectory information (i.e., the sequence of links in the plan) and ignore the storage of the expected arrival time at each link in the plan.]

Information-Theoretic Approach

- *Information is stored, or coded, as a structured collection (a* ***message****) of primitive* ***symbols****.*
- *Symbols belong to an* ***alphabet****.*
- *Messages typically have non-random statistical properties.*
  - *The frequency distribution of symbols is non-uniform.*
  - *Correlations exist between the symbols at various locations in a message.*
- *The information content of a message can be measured and the knowledge obtained about its structure can be exploited to construct better alphabets for the storage of the information.*

Link ID (ASCII)

- This method simply stores the succession of link IDs (numbers between 1 and $2^{32} - 1$) as space-delimited ASCII text.
- This is similar to the current TRANSIMS ASCII planner output file format.
- Information is stored inefficiently.
  - Many of the symbols in the alphabet (the ASCII character set) never appear in the message.
- The formatted data is easy to view/modify manually with a text editor.
- Example:

  51743400 51743508 51931500 5000005 5000012 5000013 4950508 4950509 4950510 4951804 4951805 4901500 4902003

Link ID (binary)

- *This method stores the succession of link IDs as a series of 32-bit binary words.*
- *This is similar to the current TRANSIMS binary plan file format.*
- *Information is stored inefficiently.*
  - *Many of the symbols in the alphabet (32-bit binary words) never appear in the message.*
- *The formatted data requires no special pre/post-processing by the TRANSIMS microsimulation/planner.*
- *Example:*

  $03158AA8_{16}$ $03158B14_{16}$ $0318696C_{16}$ $004C4B45_{16}$ $004C4B4C_{16}$ $004C4B4D_{16}$ $004B89EC_{16}$ $004B89ED_{16}$ $004B89EE_{16}$ $004B8EFC_{16}$ $004B8EFD_{16}$ $004ACA7C_{16}$ $004ACC73_{16}$

Unique Link ID (binary)

- *A typical network only uses a fraction of the possible link IDs (3 parts-per-million in the plan set we looked at).*
- *One can reduce the storage required for link IDs by temporarily renumbering them consecutively when they are stored.*
- *Information is stored inefficiently.*
  - *Strong correlations exist between consecutive alphabetic symbols (links) in the message.*
- *The correspondence between the original link IDs and the renumbered ones must be stored in an auxiliary table, so the format requires pre/post-processing by the TRANSIMS microsimulation/planner.*
- *Example:*

$013A_{16}$ $9B39_{16}$ 7268_{16} 0021_{16} $DD3F_{16}$ $42F7_{16}$ $8A11_{16}$ $92FA_{16}$ $73E9_{16}$ $F2F0_{16}$ 5529_{16} 6480_{16} $4AA1_{16}$

where unique link IDs map into original link IDs via $013A_{16} \rightarrow 03158AA8_{16}$, $9B39_{16} \rightarrow 03158B14_{16}$, *etc.*

ASIDE: Straight-Through Links

- *We use the following conventions for labeling turns and determining the **straight-through link**:*
  - *Consider the outgoing links at a node with allowed movements from a given incoming link.*
  - *The straight-through link is the link of the same functional class with the smallest deflection angle, provided the deflection angle is less than 60°.*
  - *If there is no outgoing link of the same functional class, then the straight-through link is the outgoing link with the smallest deflection angle, also provided the deflection angle is less than 60°.*
  - *If all outgoing links have deflection angles greater than 60°, then no straight-through link is defined.*
- *The **straight-through direction** is the angle of the straight-through link, if one is defined, or the angle of the incoming link, otherwise.*
- *This scheme is a slight variation of the definition developed by Ron Smith and Kathy Berkbigler.*

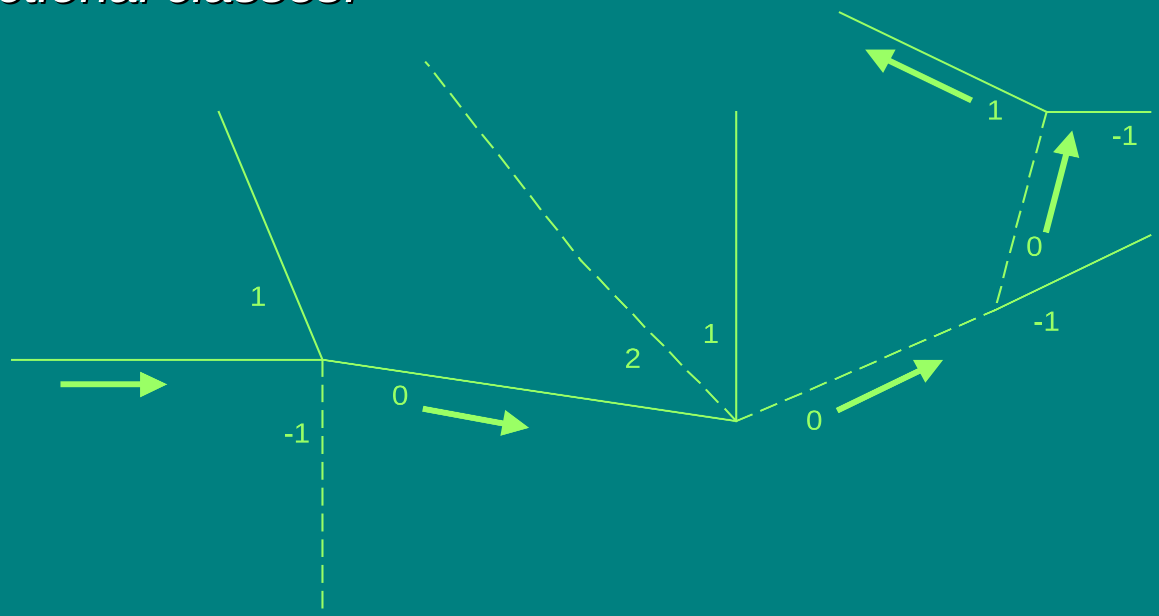
Run-Length Encoding of Link ID (ASCII)

- *Vehicles mostly travel straight through intersections: about 80% of the movements in the plan set we looked at are straight-through movements.*
- *One can reduce the number of links stored by assuming the straight-through movement as the default at an intersection and storing only non-default movements.*
- *Information is stored inefficiently.*
  - *Correlations exist between consecutive alphabetic symbols (links) in the message.*
  - *Many of the symbols in the alphabet (the ASCII character set) never appear in the message.*
- *The formatted data is possible to view/modify manually with a text editor if a map of the road network is available.*
- *Example:*

  51743400 [51743508 51931500] 5000005 [5000012 5000013 4950508 4950509 4950510] 4951804 4951805 [4901500] 4902003

ASIDE: Labeling Turns

- *Turns are numbered relative to the straight-through direction, with consecutive positive integers assigned to left turns and with consecutive negative integers assigned to right turns.*
- *The diagram below shows a path through the road network (denoted by the arrow) along with the numbering of the turns; the solid and dashed road segments represent links of different functional classes.*



Turn (ASCII)

- Path information can be encoded it by labeling the sequence of turns in the plan.
- We can assign any turn an integer value that must be interpreted in the context of the intersection and incoming link where the turn takes place.
- The integer values for turns can be represented as an ASCII sequence of digits.
- Information is stored inefficiently.
  - Many of the symbols in the alphabet (the ASCII character set) never appear in the message.
  - The frequency of the symbols is highly non-uniform.
- The correspondence between the turns and link-to-link movements must be stored in an auxiliary table, so the format requires pre/post-processing by the TRANSIMS microsimulation/planner.

Turn (ASCII) — continued

- *The formatted data is easy to view/modify manually with a text editor if a map of the road network is available.*
- *Example:*

0 0 1 0 0 0 0 0 2 -1 0 1

where turn 0 coming from link 51743400 means to go to link 51743508, turn 1 coming from link 51931500 means to go to link 5000005, etc.

Turn (binary)

- This method stores the turns in 4-bit binary half-bytes.
- Information is stored inefficiently.
  - The frequency of the symbols is highly non-uniform.
  - Correlations exist between consecutive alphabetic symbols in the message.
- The correspondence between the turns and link-to-link movements must be stored in an auxiliary table, so the format requires pre/post-processing by the TRANSIMS microsimulation/planner.
- Example:

  0000_2 0000_2 0001_2 0000_2 0000_2 0000_2 0000_2 0000_2 0010_2 1001_2 0000_2 0001_2

  where turn 0000_2 coming from link 51743400 means to go to link 51743508, turn 0001_2 coming from link 51931500 means to go to link 5000005, etc.

Huffman Turn Compression (binary)

- *Certain series of turns in a plan set are more common than other series: about 30% of the length-eight series in the plan set we looked at are eight consecutive straight-through movements.*
- *One can construct a binary representation of the series where the number of bits needed to represent a series of turns is inversely related to the frequency of the series.*
  - *Measuring the frequency of all of the turn series of a given length.*
  - *Each series is considered a symbol in the alphabet from which the plans are constructed.*
  - *The optimal binary encoding of the alphabet for the plan set is the* ***Huffman Code*** *for the alphabet.*
- *The correspondence between the binary code and the turn series must be stored in an auxiliary table, so the format requires pre/post-processing by the TRANSIMS microsimulation/planner.*

Huffman Turn Compression (binary) — theory

- *Represent the binary code as a binary tree.*
- *The alphabetic symbols in the message are at terminal nodes in the tree.*
- *The depth of a symbol is related to its relative frequency by* $d_i \sim -\log_2 f_i$.
- *The overall number of bits necessary to encode the message is* $\Sigma_i f_i d_i \sim -\Sigma_i f_i \log_2 f_i = H$, *i.e., the message's entropy.*

Huffman Turn Compression (binary) — example

- *Consider single turns (the length of the turn series is unity).*
- *Measure the frequencies of the symbols.*
- *Construct the Huffman code by combining the least-frequent pairs iteratively into a binary tree:*

  turn -3 $\rightarrow$ 010010_2

  turn -2 $\rightarrow$ 01000_2

  turn -1 $\rightarrow$ 00_2

  turn 0 $\rightarrow$ 1_2

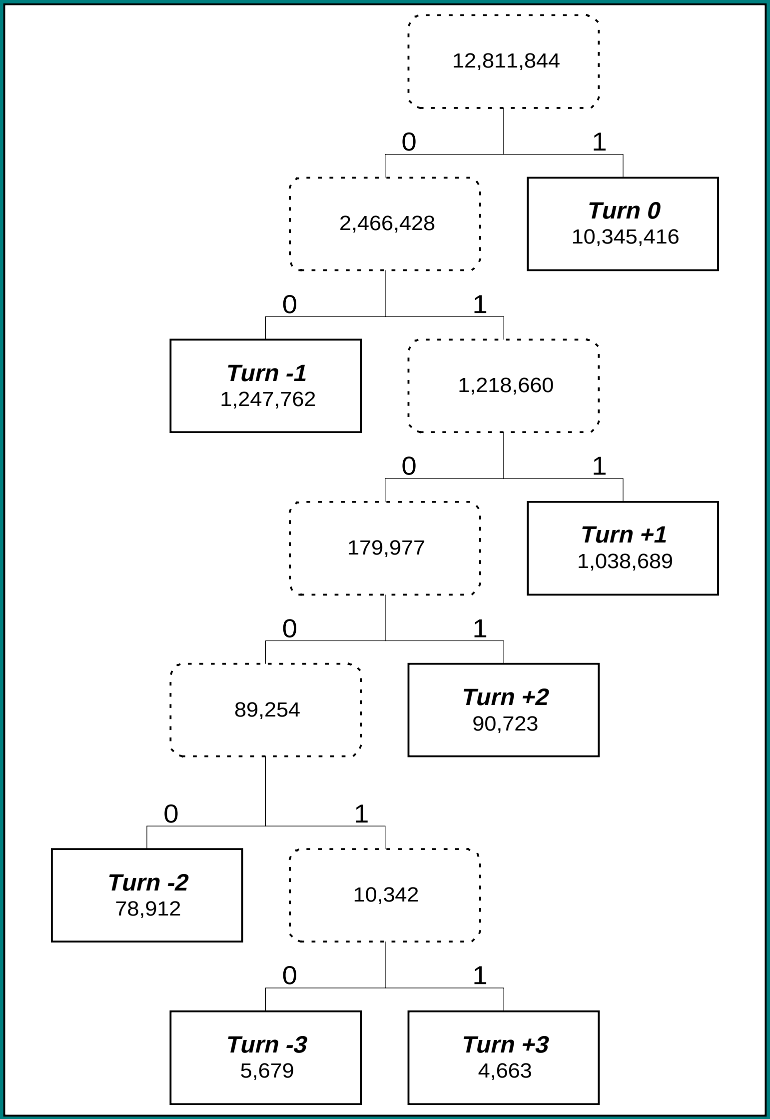
  turn +1 $\rightarrow$ 011_2

  turn +2 $\rightarrow$ 0101_2

  turn +3 $\rightarrow$ 010011_2

- *Encode the data:*

  $1_2\ 1_2\ 011_2\ 1_2\ 1_2\ 1_2\ 1_2\ 1_2\ 0101_2\ 00_2\ 1_2\ 011_2$



Other Methods

- *A slightly less optimal Huffman code can be constructed from a small sample (1%, for example) of the plan set, so that encoding can take place as plans are generated.*
- *It is probably possible to construct variations on the Huffman encoding scheme to take better account of the correlation in turn series by allowing variable-length turn series; this will reduce the number of low-frequency series in the Huffman code.*
- *It may be practical to create a separate length-one or length-two Huffman code for each intersection—so far we have only considered codes globally applicable to the whole road network.*
- *One can construct a Huffman code based on link ID.*
- *The Lempel-Ziv (LZ) algorithm can be applied to a plan set.*

Plan Set for Comparison Study

- *file: /transims/output2/segovia/PlanSets/12-25-96-k5*
- *247,620,952 bytes (ASCII)*
- *98,422,112 bytes (binary)*
- *294,702 plans*
- *13,112,498 steps*
- *12,817,796 turns*
- *14,750 links*

Comparison Study Results

Storage Method			*Estimated Plan Set Size (MB)*
Link ID (ASCII)			91.79
Link ID (binary)			52.45
Unique Link ID (binary)			22.70
Run-Length Encoding of Link ID (ASCII)			17.67
Turn (ASCII)			12.81
Turn (binary)			4.81
Huffman Turn Compression (binary)	*Series Length*	*Number of Unique Series*	
	1	7	1.56
	2	54	1.62
	3	268	1.61
	4	949	1.58
	5	2778	1.56
	6	6945	1.53
	7	14924	1.50
	8	28789	1.47

Comparison of Methods

Storage Method	*Advantages*	*Disadvantages*
Link ID (ASCII)	• viewable with text editor • link queries possible	• requires a lot of storage
Link ID (binary)	• link queries possible	• requires a lot of storage
Unique Link ID (binary)	• link queries possible	• requires moderate storage
Run-Length Encoding of Link ID (ASCII)	• eliminates simple redundancy	• requires moderate storage • link queries not possible • hard to detect invalid plans
Turn (ASCII)	• viewable with text editor	• link queries not possible
Turn (binary)	• requires little storage	• link queries not possible
Huffman Turn Compression (binary)	• requires near-minimal storage • impossible to code invalid plans	• link queries not possible • preprocessing required before storage

Future Work

- Consider additional storage methods, including “lossy” methods that discard information.
- Consolidate/package the C++ classes used for these experiments.
- Study how to store efficiently link arrival times in plans.
- Investigate whether similar methods can be applied to microsimulation output.